Docket No.: 03226/390001; P8399
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Matthew A. Ahrens et al.

Application No.: 10/828,573

Filed: April 21, 2004

For: METHOD AND SYSTEM FOR SELF-VALIDATING CHECKSUMS IN A FILE SYSTEM

Confirmation No.: 5289

Art Unit: 2135

Examiner: S. Debnath

APPEAL BRIEF

MS Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Madam:

As required under § 41.37(a), this brief is filed within two months of the Notice of Appeal filed in this case on January 9, 2009, and is in furtherance of said Notice of Appeal.

TABLE OF CONTENTS

This brief contains items under the following headings as required by 37 C.F.R. § 41.37 and M.P.E.P. § 1205.2:

TABLE OF AUTHORITIES

A. Court and Administrative Decisions

B. Statutes

C. Other Authorities

I. REAL PARTY IN INTEREST

The real party in interest for the referenced application is Sun Microsystems, Inc. An Assignment transferring all interest in the referenced application from the inventors to Sun Microsystems, Inc. was filed with the USPTO on April 19, 2004. The Assignment is recorded at Reel 015253, Frame 0026.

II. STATEMENT OF RELATED CASES

There are no other applications, appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. JURISDICTIONAL STATEMENT

The Board has jurisdiction under 35 U.S.C. § 134(a). The Examiner mailed a final rejection on October 16, 2008 ("Final Rejection"), setting a three-month shortened statutory period for response. The time for responding to the final rejection expired on January 16, 2009. Rule 134. Under 37 C.F.R. § 1.134, a notice of appeal was filed on January 9, 2009. The time for filing an appeal brief is two months after the filing of a notice of appeal. Bd. R. 41.37(c). The time for filing an

appeal brief expires on March 9, 2009. The appeal brief is being filed on March 9, 2009.

IV. STATUS OF AMENDMENTS

All amendments have been entered and considered by the Examiner. The pending claims of record are presented in the Claims Appendix.

V. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The rejections to be reviewed on this appeal include the rejection of claims 1, 5, 7-11, 18, and 20-27 under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent Pub. No. 2002/0161972 ("Talagala"), in view of U.S. Pat. No. 6,829,617 ("Sawdon") in further view of U.S. Pat. No. 6,192,471 ("Pearce").

VI. STATEMENT OF FACTS

Claims 1, 5, 7-11, 18, and 20-27 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Talagala in view of Sawdon in further view of Pearce. The claims are directed to storing and retrieving data from storage pool in a manner that the data may be self-validated. As required by the independent claims, the storage

pool is arranged in a hierarchical tree and the self-validation is performed using checksums.

A checksum is used to determine whether a piece of data has been corrupted between the time the data was stored and the time in which the data was retrieved. Specifically, a checksum is an ordered set of bits that are obtained by applying a formula to data. When the data is first stored, the formula is applied to the data to generate the checksum, and the checksum is stored. When the data is retrieved from storage, the same formula is applied to the retrieved data and a new checksum is generated. If the new checksum matches the stored checksum, then the data has remained unaltered. If the new checksum does not match the stored checksum, then the data has been corrupted. Thus, a checksum provides a mechanism for determining whether data in storage is corrupt.

In the present application, the independent claims require, in part, that a first data block and a second data block are stored on a first level of a hierarchical tree. A first indirect block is stored on a second level of the hierarchical tree. The first indirect block includes a pointer to the first data block and a pointer to the second data block. A checksum, calculated for the first data block, is stored in the first indirect block. Similarly, a checksum calculated for the second data block, is also

stored in the first indirect block. Thus, the first indirect block includes the following: (i) a pointer to the first data block, (ii) a pointer to the second data block, (iii) a checksum calculated for the first data block, and (iv) a checksum calculated for the second data block. Moreover, the independent claims require, in part, that a checksum is calculated for the first indirect block. The checksum for the first indirect block and a pointer to the first indirect block is stored in a second indirect block. The second indirect block is stored on a third level of the hierarchical tree. *See*, *e.g.*, Figures 3-6, p. 8, ll. 16-29, p. 10, ll. 22 - p. 11, ll. 16 of the application as filed.

Turning to the prior art relied on by the Examiner, the Examiner cites Talagala, Sawdon, and Pearce. *See* Office Action dated October 16, 2008, p. 2-4. Talagala is directed to validating and correcting data in a data storage array. *See*, *e.g.*, Talagala, abstract. To validate the data, Talagala teaches using checksums. As disclosed in Talagala, to recover the original uncorrupted parity blocks are used. *See*, *e.g.*, Talagala, paragraph [0006], [0008], and [0011].

The use of parity blocks is a well-known concept in computer science for recovering original uncorrupted data once data corruption is detected. Parity blocks are used as follows. When a storage system has multiple storage devices, data may be divided into multiple data blocks and striped across the storage devices. In

particular, each of the data blocks in the stripe is stored on a separate storage device. A final data block in the stripe, called a parity block, is created from the data blocks. The parity block is created by performing a bitwise XOR function on the data blocks. The data blocks and the parity block form a parity group. The use of the bitwise XOR function for the parity block allows for easy recovery of any of the data blocks in the parity group. Specifically, if corrupt data is detected and the corrupted data block is identified (*e.g.*, by using a checksum associated with the data block), the original data in the single data block may be recovered by using the bitwise XOR function of the remaining data blocks in the parity group. However, when a data block in the parity group is corrupt, the parity block itself cannot be used to identify which data block is corrupt. *Id.*

As a simple example, if there are four storage devices, then a parity group contains three data blocks and a parity block. In the simple example, consider the scenario in which the data blocks are 0011, 0100, and 1010. In such an example, the parity block is 1101 (i.e., the bitwise XOR of 0011, 0100, and 1010). In the example, consider the scenario in which the first data block (i.e., 0011) is corrupt. Specifically, the first data block may store 1011 rather than the correct data of 0011. As discussed above, the corruption may be detected by calculating the checksum of

1011, and comparing the calculated checksum with the stored checksum that was calculated for 0011. Because 1011 and 0011 are not the same, the stored checksum would not match the calculated checksum.

Thus, in the example, the correct data for the data block is recovered. To recover the original uncorrupt data for the first data block, a bitwise XOR function is performed on the remaining data blocks (i.e., 0100, 1010, and 1101) in the parity group. The result of the XOR function is 0011 (i.e., the original first data block). As shown by way of example, although parity blocks are related to checksums in that both are used for correcting errors in stored data, checksums and parity blocks are not equivalent. Checksums are used to identify a data block that is corrupt, while parity blocks are used to recover the original data once the corrupted data block is identified.

Returning to Talagala, Talagala addresses the problem of updating the parity block when a data block is purposefully modified. Specifically, when data in a data block is modified, a new data block with the modified data is stored in a new location on the same storage device. The parity group of the old data block is identified. The new data block replaces the old data block in the parity group. A new

parity block is calculated for the updated parity group and is also stored. *See, e.g.*, Talagala, Figure 4 and paragraph [0016], and [0043] – [0045].

To keep track of the updated data blocks in the parity group, Talagala teaches maintaining a hash indirection table and a parity group table. The hash indirection table maps a virtual addresses of the data blocks and parity blocks to entries in the parity group table. The parity group table is a data structure. The parity group table is stored separately from the data blocks, and includes an entry for each data block. *See, e.g.*, Talagala, Figures 6B-6C and paragraphs [0055]-[0058].

An entry for the data block in the parity group table identifies the following: (1) the segment, (2) the type of data block (*i.e.*, does not contain data, contains data, or a parity block), (3) a checksum for the data block, and (4) the next data block in the parity group. Talagala discloses that the identification of the next data block in the parity group forms a cycle. Specifically, the entry for the data block on the first storage device references the entry for the data block on the second storage device, which references the entry for the data block on the third storage device, etc. The entry for the data block on the last storage device references the entry for the data block on the first storage device. *Id.*

Turning to the rejection, the Examiner relies on the parity group table and the data blocks to support the rejection that Talagala discloses (1) a first indirect block, which includes a pointer to the first data block, a pointer to the second data block, a checksum calculated for the first data block, and a checksum calculated for the second data block, and (2) a second indirect block, which has a pointer to the first indirect block and a checksum for the first indirect block. *See* Office Action dated October 16, 2008, p. 2-3.

To further support the rejection of the independent claims, the Examiner relies on Sawdon to teach that the first data block, the second data block, the first indirect block, and the second indirect block are located in separate physical locations. *See* Office Action dated October 16, 2008, p. 3-4.

Additionally, the Examiner relies on Pearce to teach that the first data block and second data block are stored on a first level of a hierarchical tree, the first indirect block is stored on a second level of a hierarchical tree, and the second indirect block is stored on a third level of a hierarchical tree. *See* Office Action dated October 16, 2008, p. 4. Pearce teaches that the hierarchical tree is used to track which segments of memory do not contain data. *See, e.g.*, Pearce, Figure 2C and col. 4, ll.

53-57. Namely, the hierarchical tree in Pearce is used to identify which segments of memory may be used to store data.

VII. ARGUMENT

The arguments presented below have not previously been presented to the Examiner. In this Appeal, the Appellants assert that the Examiner erred in contending that claims 1, 5, 7-11, 18, and 20-27 under 35 U.S.C. § 103(a) as being unpatentable over Talagala in view of Sawdon in further view of Pearce. For the purpose of this appeal, claims 1, 5, 7-11, 18, and 20-27 stand or fall together. Claim 1 is representative of the group containing claims 1, 5, 7-11, 18, and 20-27. Accordingly, the rejection is respectfully traversed.

MPEP §2143 states that "[t]he key to supporting any rejection under 35 U.S.C. 103 is the clear articulation of the reason(s) why the claimed invention would have been obvious." Further, the Supreme Court in *KSR International Co. v. Teleflex Inc.*, 127 S.Ct. 1727, 1739, 75 U.S.L.W. 4289 (2007) requires that the Examiner "articulate the following: (1) a finding that the prior art included each element claimed, although not necessarily in a single prior art reference, with the only difference between the claimed invention and the prior art being the lack of actual

combination of the elements in a single prior art reference; ...." MPEP § 2143(A) citing *KSR International Co. v. Teleflex Inc.*, 127 S.Ct. 1727, 1739, 75 U.S.L.W. 4289 (2007). Under MPEP § 2141.02, when interpreting the claim, "[d]istilling an invention down to the "gist" or "thrust" of an invention disregards the requirement of analyzing the subject matter "as a whole."" *See* MPEP 2141.02 citing *W.L. Gore & Associates, Inc. v. Garlock, Inc.*, 721 F.2d 1540, 220 USPQ 303 (Fed. Cir. 1983), cert. denied, 469 U.S. 851 (1984) ("restricting consideration of the claims to a 10% per second rate of stretching of unsintered PTFE and disregarding other limitations resulted in treating claims as though they read differently than allowed").

In support of the rejection of claim 1, the Examiner ignores limitations of claim 1 that require that a first indirect block includes checksums for at least two data blocks and has a checksum calculated for it in a second indirect block. Specifically, in support of the rejection of claim 1, the Examiner relies on the portion of Talagala that is directed to a parity group table. The Examiner merely references entries in the parity group table and the circular relationships between entries to teach the aforementioned limitation without referencing how Talagala teaches or suggests a first indirect block includes checksums for at least two data blocks. In other words, the Examiner merely identifies a table that has a general concept of checksum and a general concept of references between entries in a table without due consideration to

the structure required by the aforementioned limitation of claim 1. Not only does Talagala fail to teach or suggest the aforementioned limitation, but also such assertion by the Examiner attempts to distill the claim to the gist of the invention while disregarding the requirement of analyzing the subject matter as a whole. Therefore, such assertion is wholly improper under *W.L. Gore & Associates, Inc.*

In fact, as will be shown below, Talagala is completely silent with respect to a first indirect block that (1) stores a checksum for a first data block and a checksum for a second data block and (2) has a checksum stored for it in a second indirect block. The following discussion shows how (i) the parity group table as a whole cannot be equated to the first indirect block; (ii) each entry in the parity group table cannot correspond to an indirect block with one of the entries being the first indirect block; and (iii) the entries of the parity group table cannot be equated to data blocks and the indirect blocks.

The parity group table as a whole cannot be equated to the first indirect block

The parity group table as a whole cannot be equated to the first indirect block. Specifically, assume, *arguendo*, that the parity group table may be equated to a first indirect block. In such scenario, claim 1 requires that a checksum for the first indirect block is stored in a second indirect block, claim 1 requires that a checksum is

calculated and stored for the parity group table itself in a second indirect block. However, Talagala is completely silent with respect to calculating a checksum for the parity group table. Thus, if the parity group table is equivalent the first indirect block, then Talagala fails to teach or suggest that a checksum is calculated for the first indirect block and stored in the second indirect block as required by claim 1. Because the parity group table fails to have the properties of the first indirect block required by claim 1, the parity group table cannot be equated to the first indirect block.

Each entry in the parity group table does not correspond to an indirect block with one of the entries being the first indirect block

Further, each entry in the parity group table does not correspond to an indirect block with one of the entries being the first indirect block. Specifically, assume, *arguendo*, that each entry in the parity group table corresponds to an indirect block, and one of the entries is the first indirect block. However, Talagala fails to teach or suggest that a checksum is calculated for the entry of the table itself. Specifically, Talagala discloses that checksums are only calculated for the data block corresponding to the entry. Because Talagala fails to teach or suggest that a checksum is calculated for the entry itself and claim 1 requires that a checksum calculated for the first indirect block, an entry cannot be equated to a first indirect block.

Moreover, each entry only stores a checksum for a single data block. Because claim 1 requires that the first indirect blocks stores a checksum for at least two data blocks, an entry in the parity group table which only stores a checksum for a single data block cannot be equated to the first indirect block. Therefore, an entry, by itself, is not a first indirect block.

The entries of the parity group table cannot be equated to data blocks and the indirect blocks

Finally, the entries of the parity group table cannot be equated to both data blocks and the indirect blocks. Specifically, assume, *arguendo*, that the entries themselves are the data blocks and the indirect blocks. As discussed above, the claim 1 requires that a checksum is calculated for an indirect block. However, Talagala fails to teach or suggest that a checksum is calculated and stored for the entry itself as would be required by the claims of the present invention. Therefore, even if the entries were able to be equated to the data blocks and indirect blocks of claim 1, Talagala still fails to teach or suggest that a checksum is calculated and stored for one of the entries as would be required by claim 1. In view of the above, the entries of the parity group table cannot be equated to both data blocks and the indirect blocks.

Summary

As shown above, neither the parity group table as a whole nor the entries of the parity group table can be used to teach or suggest a first indirect block. In fact, nothing in the parity group table of Talagala can be equated to a first indirect block having checksums for at least two data blocks, and a checksum stored for it in a second indirect block. Namely, in contrast to the Examiner's broad assertions, Talagala is silent with respect to the first indirect block with the properties required by claim 1.

Moreover, Sawdon and Pearce fail to teach or suggest that which Talagala lacks. Specifically, both Sawdon and Pearce are silent with respect to checksums. Therefore, neither Sawdon and Pearce can teach a first indirection block have checksums for at least two data blocks, and that a checksum for the first indirection block is stored in a second indirection block as required by claim 1.

In view of the above, the Examiner has failed to show the presence of all elements in the prior art and thereby has failed to satisfied the requirements of *KSR International Co.,* which requires that the Examiner "articulate the following: (1) a finding that the prior art included each element claimed, although not necessarily in a single prior art reference, with the only difference between the claimed invention and the prior art being the lack of actual combination of the elements in a single prior art

reference; …" MPEP § 2143(A) citing *KSR International Co. v. Teleflex Inc.*, 127 S.Ct. 1727, 1739, 75 U.S.L.W. 4289 (2007). Accordingly, the Examiner has failed to show sufficient evidence of *prima facie* obviousness for claims 1-3, 5-17, 19-29, and 31-36.

VIII. CONCLUSION

In view of the above, as the Examiner has failed to show sufficient evidence for a *prima facie* to support prima facie obviousness, the Appellants have carried their burden in showing that the Examiner erred in finally rejecting the claims. *In re Kahn*, 441 F.3d 977, 985-986 (Fed. Cir. 2006) ("On appeal to the Board, an applicant can overcome a rejection by showing insufficient evidence of *prima facie* obviousness or by rebutting the *prima facie* case with evidence of secondary indicia of nonobviousness") (emphasis in original) (quoting *In re Rouffet*, 149 F.3d 1350, 1355 (Fed. Cir. 1998)); *see also* 37 C.F.R. § 41.37(c)(1)(vii). Favorable consideration of the present application is respectfully requested.

Dated: March 9, 2009

Respectfully submitted,

By /Robert P. Lord/

Robert P. Lord
Registration No.: 46,479
OSHA · LIANG LLP
909 Fannin Street, Suite 3500
Houston, Texas 77010
(713) 228-8600
(713) 228-8778 (Fax)

APPENDIX A - CLAIMS

1. (Rejected) A method for storing data blocks, comprising:

storing a first data block and a second data block in a storage pool, wherein data stored in the storage pool is organized as a hierarchical tree, and wherein the first and second data block are stored on a first level of the hierarchical tree;

obtaining a first data block location and a second data block location;

calculating a first data block checksum for the first data block;

calculating a second data block checksum for the second data block;

storing a first indirect block at a first indirect block location on a second level of the hierarchical tree in the storage pool, wherein the first indirect block comprises a first block pointer comprising the first data block location and the first data block checksum and a second block pointer comprises the second data block location and the second data block checksum;

calculating a first indirect block checksum for the first indirect block by applying a checksum function to the first indirect block; and

storing a second indirect block at a second indirect block location on a third level of the hierarchical tree in the storage pool, wherein the second indirect block comprises the first indirect block location and the first indirect block checksum,

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool.

2. – 4. (Cancelled)

5. (Rejected) The method of claim 1, further comprising:
storing a birth value in a birth field in the first block pointer.

6. (Cancelled)

7. (Rejected) The method of claim 1, wherein the storage pool comprises at least one storage device.

8. (Rejected) The method of claim 1, wherein the storage pool is divided into a plurality of metaslabs.

9. (Rejected) The method of claim 8, wherein each of the plurality of metaslabs is associated with a metaslab ID.

10. (Rejected) The method of claim 9, wherein the first data block location comprises the metaslab ID and an offset.

11. (Rejected) The method of claim 1, wherein storing the first data block and the second data block comprises using a storage pool allocator.

12.– 17 (Cancelled)

18. (Rejected) A system for storing data blocks, comprising:
a storage pool configured to store data, wherein data stored in the storage pool is organized as a hierarchical tree, comprising:
a first data block, a second data block;
a first indirect block referencing the first data block and the second data block, wherein the first indirect block comprises a first data block

checksum and a first data block location stored in a first block pointer, and a second data block checksum and a second data block location stored in a second block pointer;

a second indirect block, comprising a first indirect data block checksum and a first indirect block location; and

a storage pool allocator configured to store the first data block and the second data block on a first level of the hierarchical tree, the first indirect block on a second level of the hierarchical tree, and the second indirect block on a third level of the hierarchical tree in the storage pool,

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool.

19. (Cancelled)

20. (Rejected) The system of claim of claim 18, further comprising:

a data management unit configured to assemble the first indirect block and request the storage pool allocator to store the first indirect block.

21. (Rejected) The system of claim 20, wherein the storage pool comprises at least one storage device.

22. (Rejected) The system of claim 20, wherein the storage pool is divided into a plurality of metaslabs.

23.(Rejected) The system of claim 22, wherein each of the plurality of metaslabs is associated with a metaslab ID.

24. (Rejected) The system of claim 23, wherein the first data block location comprises the metaslab ID and an offset.

25. (Rejected) A computer system for storing data blocks, comprising:

a processor;

a memory;

a storage device; and

software instructions stored in the memory for enabling the computer system under control of the processor, to:

store a first data block and a second data block in a storage pool, wherein data stored in the storage pool is organized as a hierarchical tree, and wherein the first and second data block are stored on a first level of the hierarchical tree;

obtain a first data block location and a second data block location;

calculate a first data block checksum for the first data block;

calculate a second data block checksum for the second data block;

store a first indirect block at a first indirect block location on a second level of the hierarchical tree in the storage pool, wherein the first indirect block comprises a first block pointer comprising the first data block location and the first data block checksum and a second block pointer comprises the second data block location and the second data block checksum;

calculate a first indirect block checksum for the first indirect block by applying a checksum function to the first indirect block; and

store a second indirect block at a second indirect block location on a third level of the hierarchical tree in the storage pool, wherein the second indirect

block comprises the first indirect block location and the first indirect block checksum,

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool.

26.(Rejected) A network system having a plurality of nodes, comprising:

a storage pool, configured to store data, wherein data stored in the storage pool is organized in a hierarchical tree, comprising:

a first data block, a second data block,

a first indirect block referencing the first data block and the second data block, wherein the first indirect block comprises a first data block checksum and a first data block location stored in a first block pointer, and a second data block checksum and a second data block location stored in a second block pointer;

a second indirect block, comprising a first indirect data block checksum and a first indirect block location; and

a storage pool allocator configured to store the first data block and the second data block on a first level of the hierarchical tree, the first indirect block on a second level of the hierarchical tree, and the second indirect block on a third level of the hierarchical tree in the storage pool,

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool,

wherein the storage pool is located on any one of the plurality of nodes, and

wherein the storage pool allocator is located on any one of the plurality of nodes.

27. (Rejected) A method for retrieving data in a first data block, comprising:

obtaining a first indirect block comprising a second indirect block location and a second indirect block checksum, wherein the first indirect block is stored at a first indirect block location on a third level of a storage pool, wherein data stored in the storage pool is organized as a hierarchical tree;

obtaining the second indirect block from a second level of the hierarchical tree using the second indirect block location, wherein the second indirect block comprises a first data block pointer and a second data block pointer, wherein the first data block pointer comprises a first data block location and a first data block checksum, and the second data block pointer comprises a second data block location and a second data block checksum, wherein the first and second data block locations are on a first level of the hierarchical tree;

applying a checksum function to the second indirect block to obtain a first calculated checksum,

comparing the first calculated checksum with the second indirect block checksum;

obtaining the first data block using the first data block location when the first calculated checksum equals the second indirect block checksum;

applying the checksum function to the first data block to obtain a second calculated checksum;

comparing the second calculated checksum with the first data block checksum; and

retrieving the data from the first data block when the second calculated checksum equals the first data block checksum,

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool.

APPENDIX B – CLAIM SUPPORT AND DRAWING ANALYSIS

The following references to the Figures and Specification referenced below should not be construed as the only locations in the specification which support or discuss of the respective limitation.

1. A method for storing data blocks, comprising: **{p. 3, ll. 5-8}**

storing a first data block and a second data block in a storage pool, wherein data stored in the storage pool is organized as a hierarchical tree, and wherein the first and second data block are stored on a first level of the hierarchical tree; **{Figure 3; Figure 4: 106; p. 8, ll. 16-21}**

obtaining a first data block location and a second data block location; **{Figure 4:108; p. 8, ll. 16-21}**

calculating a first data block checksum for the first data block; **{Figure 4: 108; p. 10, ll. 22-p. 11, ll. 5}**

calculating a second data block checksum for the second data block; **{Figure 4: 108; p. 10, ll. 22-p. 11, ll. 5}**

storing a first indirect block at a first indirect block location on a second level of the hierarchical tree in the storage pool, wherein the first indirect block comprises a first block pointer comprising the first data block location and the first data block checksum and a second block pointer comprises the second data block location and the second data block checksum; **{Figure 4: 112, 116; p. 8, ll. 24-29; p. 11, ll. 6-14}**

calculating a first indirect block checksum for the first indirect block by applying a checksum function to the first indirect block; and **{Figure 4: 120; p. 11, ll. 6-14}**

storing a second indirect block at a second indirect block location on a third level of the hierarchical tree in the storage pool, wherein the second indirect block comprises the first indirect block location and the first indirect block checksum, **{Figure 4: 112, 116, 120; Figure 5; p. 8, ll. 24-29; p. 11, ll. 15-29}**

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool. **{p. 8, ll. 16-18}**

18. A system for storing data blocks, comprising: **{p. 4, ll. 7-11}**

a storage pool configured to store data, wherein data stored in the storage pool is organized as a hierarchical tree, comprising: **{Figure 1: 108; p. 8, ll. 16-21}**

a first data block, a second data block; **{p. 8, ll. 16-21}**

a first indirect block referencing the first data block and the second data block, wherein the first indirect block comprises a first data block checksum and a first data block location stored in a first block pointer, and a second data block checksum and a second data block location stored in a second block pointer; **{Figure 5; p. 11, ll. 15-29}**

a second indirect block, comprising a first indirect data block checksum and a first indirect block location; and **{Figure 5; p. 11, ll. 15-29}**

a storage pool allocator configured to store the first data block and the second data block on a first level of the hierarchical tree, the first indirect block on a second level of the hierarchical tree, and the second indirect block on a third level of the hierarchical tree in the storage pool, **{Figure 1: 106; p. 10, ll. 22 – p. 11, ll. 14}**

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool. **{p. 8, ll. 16-18}**

25. A computer system for storing data blocks, comprising: **{p. 4, ll. 12-18}**

a processor; **{Figure 7; p. 12, ll. 21-25}**

a memory; **{Figure 7; p. 12, ll. 21-25}**

a storage device; and **{Figure 7; p. 12, ll. 21-25}**

software instructions stored in the memory for enabling the computer system under control of the processor, to: **{p. 4, ll. 12-18}**

store a first data block and a second data block in a storage pool, wherein data stored in the storage pool is organized as a hierarchical tree, and wherein the first and second data block are stored on a first level of the hierarchical tree; **{Figure 3; Figure 4: 106; p. 8, ll. 16-21}**

obtain a first data block location and a second data block location; **{Figure 4:108; p. 8, ll. 16-21}**

calculate a first data block checksum for the first data block; **{Figure 4: 108; p. 10, ll. 22-p. 11, ll. 5}**

calculate a second data block checksum for the second data block; **{Figure 4: 108; p. 10, ll. 22-p. 11, ll. 5}**

store a first indirect block at a first indirect block location on a second level of the hierarchical tree in the storage pool, wherein the first indirect block comprises a first block pointer comprising the first data block location and the first data block checksum and a second block pointer comprises the second data block location and the second data block checksum; **{Figure 4: 112, 116; p. 8, ll. 24-29; p. 11, ll. 6-14}**

calculate a first indirect block checksum for the first indirect block by applying a checksum function to the first indirect block; and **{Figure 4: 120; p. 11, ll. 6-14}**

store a second indirect block at a second indirect block location on a third level of the hierarchical tree in the storage pool, wherein the second indirect block comprises the first indirect block location and the first indirect block checksum, **{Figure 4: 112, 116, 120; Figure 5; p. 8, ll. 24-29; p. 11, ll. 15-29}**

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool. **{p. 8, ll. 16-18}**

26. A network system having a plurality of nodes, comprising: **{p. 4, ll. 19-25; p. 12, ll. 25 – p. 13, ll. 5}**

a storage pool, configured to store data, wherein data stored in the storage pool is organized in a hierarchical tree, comprising: **{Figure 1: 108; p. 8, ll. 16-21}**

a first data block, a second data block, **{p. 8, ll. 16-21}**

a first indirect block referencing the first data block and the second data block, wherein the first indirect block comprises a first data block checksum and a first data block location stored in a first block pointer, and a second data block checksum and a second data block location stored in a second block pointer; **{Figure 5; p. 11, ll. 15-29}**

a second indirect block, comprising a first indirect data block checksum and a first indirect block location; and **{Figure 5; p. 11, ll. 15-29}**

a storage pool allocator configured to store the first data block and the second data block on a first level of the hierarchical tree, the first indirect block on a second level of the hierarchical tree, and the second indirect block on a third level of the hierarchical tree in the storage pool, **{Figure 1: 106; p. 10, ll. 22 – p. 11, ll. 14}**

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool, **{p. 8, ll. 16-18}**

wherein the storage pool is located on any one of the plurality of nodes, and **{p. 12, ll. 25 – p. 13, ll. 5}**

wherein the storage pool allocator is located on any one of the plurality of nodes. **{p. 12, ll. 25 – p. 13, ll. 5}**

27. A method for retrieving data in a first data block, comprising: **{p. 3; ll. 19-25}**

obtaining a first indirect block comprising a second indirect block location and a second indirect block checksum, wherein the first indirect block is stored at a first indirect block location on a third level of a storage pool,

wherein data stored in the storage pool is organized as a hierarchical tree; **{Figure 5; p. 11, ll. 15-29; p. 12, ll. 3-5}**

obtaining the second indirect block from a second level of the hierarchical tree using the second indirect block location, wherein the second indirect block comprises a first data block pointer and a second data block pointer, wherein the first data block pointer comprises a first data block location and a first data block checksum, and the second data block pointer comprises a second data block location and a second data block checksum, wherein the first and second data block locations are on a first level of the hierarchical tree; **{Figure 5; p. 11, ll. 15-29; p. 12, ll. 3-5}**

applying a checksum function to the second indirect block to obtain a first calculated checksum, **{Figure 6: 146; p. 12, ll. 7-8}**

comparing the first calculated checksum with the second indirect block checksum; **{Figure 6: 148; p. 12, ll. 8-9}**

obtaining the first data block using the first data block location when the first calculated checksum equals the second indirect block checksum; **{Figure 6: 150, 152; p. 12, ll. 12-20}**

applying the checksum function to the first data block to obtain a second calculated checksum; **{Figure 6: 148; p. 12, ll. 8-20}**

comparing the second calculated checksum with the first data block checksum; and **{Figure 6: 148-158; p. 12, ll. 8-20}**

retrieving the data from the first data block when the second calculated checksum equals the first data block checksum, **{Figure 6: 152; p. 12, ll. 13-15}**

wherein each of the first data block location, the second data block location, the first indirect block location, and the second indirect block location are separate physical locations in the storage pool. **{p. 8, ll. 16-18}**

APPENDIX C – MEANS OR STEP PLUS FUNCTION ANALYSIS

The pending claims do not include any claims that require means or step plus function analysis.

APPENDIX D – EVIDENCE

No evidence pursuant to §§ 1.130, 1.131, or 1.132 or entered by or relied upon by the Examiner is being submitted.

APPENDIX E – RELATED CASES

No related proceedings are referenced in II. above, hence copies or decisions in related proceedings are not provided.